Lab: Supplying Inputs to Programs

May 25, 2013

Contents

In this lab, we will look at some of the different ways that basic I/O information can be supplied to a running program.

Probably most of the programs you have written and tested so far have been text-interactive. They were designed to print out a prompt on the screen (`cout`) requesting a single piece of data. Then they would read that info from the keyboard (`cin`). Then they would prompt for another piece of input, read another, and so on.

```
cout << "Enter a number between 1 and 10: "
     << flush;
int k;
cin >> k;
while (k < 1 || k > 10)
  {
   cout << "Try again: "
        << flush;
   cin >> k;
  }
cout << "Thanks!" << endl;
```

```
Enter a number between 1 and 10: 12
Try again: 9
Thanks!
```

It's rare for a useful program to actually work that way. If we really want interaction with the human operator, we would expect some sort of Graphic User Interface (GUI) with windows, pop-up dialog boxes, etc. We're not ready to start writing in that style yet. On the other hand, a lot of very useful programs are not designed to be interactive at all. They take their inputs from files, from other programs, and from parameters specified when the program is launched, and they send their output to files or other programs.

If you are going to write these kinds of programs, you need to know how to control their inputs and outputs so that you can execute and test them.

1 Running the Program

1. Run Code::Blocks and create a new project named "io_lab" with the following function:

```
#include <iostream>
#include <string>

using namespace std;

int main (int argc, char** argv) {
   cout << "Now reading from standard in" << endl;
   string line;
   getline (cin, line);
   while (cin)
     {
      cout << "I saw: " << line << endl;
```

```
      getline (cin, line);
    }
  return 0;
}
```

Compile. Run the program using the Code::Blocks blue arrow icon. When the program says it is reading from standard in, type some lines of text. Watch as it echoes your input.

This program reads until it encounters the end of input (a.k.a. end of file). How do you indicate this when you are typing?

- In Windows, type Ctrl-Z.
- In Unix/Linux, type Ctrl-D.

In both cases, you should really only do this at the start of a new line of input.

2. Now we are going to look at a few other ways to launch that program. Open up Windows Explorer ("My Computer") and navigate to the folder where you have stored this project. You should see a `bin` folder and, inside that, a `Debug` folder. In there, you will find your executable, `io_lab.exe`. Double-click on that to run the program. A command window will open in which you can interact with the program. It will close when you signal end of input.

3. Now right-click on that file and drag it somewhere (a convenient folder, or your desktop). You should see an option to create a shortcut. Choose that. Double-click the shortcut to launch the program.

   Leave that shortcut and the folder showing your executable program where you can get to them easily. We'll be using them again later.

4. Click the Windows Start button, select "Run...", and type "cmd". This opens a command window. `cd` to the folder where your executable resides.

   - The `cd` and `pwd` commands work the same way in Windows as they do in Unix (and you *did* do the required assignment in CS252 before today's lab, right?), except that Windows uses a backslash \ instead of a forward slash / to separate directories.
   - The Windows command `dir` is the closest equivalent of the Unix `ls`.
   - To switch to a different drive in Windows, give a command consisting of the drive letter followed by a colon. For example, if you stored your project on a USB flash drive that installed itself as drive E, the command is

     ```
     E:
     ```

   Once you have cd'd all the way to the directory containing your executable, you can run it by typing the file name: `lab2.exe`. Try that.

2 Supplying Standard Input

When you are testing and debugging a program, you often want to run the same input through it, over and over, as you work to fix the bugs. This can be rather tiresome if you have to actually type it out, over and over each time. It's also easy to make mistakes so that you don't quite run the tests the way you intended.

1. Run the program again, using any of the launch techniques we tried earlier. This time, instead of typing your input, copy a few lines of text from this web page (or from any file you have handy.) to the clipboard. Paste it to your running program - left click on the small box in the upper-left corner of the command window, then select `edit -> paste`.

   In practice, you can use any text editor to create files of test input data, then use copy-and-paste to feed them to your program. This includes both NotePad and, perhaps more conveniently, the Code::Blocks editor, which can certainly be used to create text files with names that don't end in `.h` or `.cpp`.

   Use the end-of-input character to close out the program.

2. Using the Code::Blocks editor, or notepad, or any convenient text editor, create a plain text file `testdata.txt` in the same directory where you have your executable. Be sure you use a text editor, not a word processor. We want a file of plain text, not the binary codes that word processors use to store fancy formatting. It doesn't matter what you put into the file - your name and address, a few lines from your favorite song, whatever. Make sure that you hit enter/return at the end of the last line.

   OK, now let's say we want to use the text in that file as input to test our program We could use the copy-and-paste technique. But when we are running from the command line, there's another option. Go to your cmd window and run your program like this:

   ```
   lab2 < testdata.txt
   ```

   You should see the program print out all the text from your file. The < character in the command is used to request **redirection** of the input. The program is still reading from `cin`, but now the data going to `cin` is being supplied from the file named after the < instead of from the keyboard.

   Redirection is a technique used heavily in Unix, and is covered in more detail in CS252. But even in Windows, it is *very* useful for testing programs that read large amounts of data from standard in.

3 Command Line Parameters

The `main` function receives two parameters. These are traditionally called `argc` and `argv`, although those aren't the most descriptive of names. These are used to supply command-line options to the program.

1. Let's add a few lines to the program:

```
#include <iostream>
#include <string>

using namespace std;

int main (int argc, char** argv)
{
   cout << "This program received " << argc << " command parameters." << endl;
   for (int i = 0; i < argc; ++i)
     {
       cout << "argv[" << i << "] is " << argv[i] << endl;
     }
    cout << "Now reading from standard in" << endl;
   string line;
   getline (cin, line);
   while (cin) {
       cout << "I saw: " << line << endl;
       getline (cin, line);
   }
   return 0;
}
```

Compile and run the program. Note the new output.

`argv` is an array of C-strings (character arrays). Each item in this array is one item from the command-line. `argc` tells you how many things are in this array.

2. Go to your cmd window and run the program like this:

```
io_lab a b cdef
```

Notice that your command line has 4 items (including the name of the program itself), and the output shows that the `argv` array has the same 4 items (the program name might be altered slightly, but the others are given exactly).

Stop the program and run it again like this:

```
io_lab "a b" cdef
```

Notice that the quotes can be used to group together a string that includes blanks, turning it into a single entry in the command parameter array.

3. A **lot** of Windows programs accept command line parameters, though you may not realize this if you always launch them by double clicking on shortcuts or selecting them from the Start menu.

Go to any empty spot on your desktop, right-click and select New->Shortcut. For the "location of the item", enter:

```
explorer
```

Click Next, give the shortcut any convenient name, and click Finish. Now double-click on your new shortcut to run it. "`explorer`" is the program that Windows uses to show you your folders. By default, it shows the folder "`My Documents`". But we can change that with a command-line parameter. Right-click on your shortcut, select properties, and to the "Target" line add a space and then the characters "c:". Save this and then run your shortcut again. Notice that the command parameter ("c:") controls which folder is displayed.

The lesson here is that even Windows programs that will probably never be invoked directly from a command line will, nonetheless, often benefit from the ability to handle command-line parameters.

Now let's return to the shortcut you created earlier. Right-click on the shortcut and select "properties". The "target" entry holds the same thing you might have typed in the cmd window. Try appending some command line parameters onto the end, click OK to save the changes, then launch the program by double-clicking the shortcut. The output should reflect your new command parameters.

4. When you run your own programs from inside Code::Blocks, you can also supply command-line parameters. This is particularly important if you want to use its built-in debugger to find bugs in a program that is designed to take information from the command line.

   In the Code::Blocks Project menu, select "Select programs' arguments...". In the "Program Arguments" box, enter a few lines of text. Run the program. You'll see that each line is treated as one command line parameter.

5. Some programs get all their input directly from their command line parameters. For example, the `expr` program on most Unix systems works like a calculator:

   ```
   > expr 1 + 4
   5
   >
   ```

   But one of the most common uses of command-line parameters is to supply file names to programs so that it will know from where to take its inputs and put its outputs. For example, change the `io_lab` program to

   ```
   #include <iostream>
   #include <fstream>
   #include <string>

   using namespace std;
   ```

```
int main (int argc, char** argv)
{
   ifstream input (argv[1]);
   ofstream output (argv[2]);
   cout << "This program received " << argc << " command parameters." << endl;
   for (int i = 0; i < argc; ++i)
     {
       cout << "argv[" << i << "] is " << argv[i] << endl;
     }
   cout << "Now reading from " << argv[1] << endl;
   cout << " and writing to " << argv[2] << endl;
   string line;
   getline (input, line);
   while (input) {
       output << "I saw: " << line << endl;
       getline (input, line);
   }
   return 0;
}
```

In your cmd window, run the program as

```
io_lab testdata.txt output.txt
```

Then open output.txt in any text editor (including the Code::Blocks editor) to view the output.

As you progress through the semester, you will be working on programs that take their inputs from a variety of sources. Try to remember that you have many options on how you supply your test inputs, whether you work entirely inside Code::Blocks or work directly at the command line.